Architecture-Based Refinements for Secure Computer Systems Design

Jie Zhou and Jim Alves-Foss[1]
Center for Secure and Dependable Systems
University of Idaho

Abstract— **The successful design and implementation of secure systems must occur from the beginning. A component that must process data at multiple security levels is very critical and must go through additional evaluation to ensure the processing is secure. It is common practice to isolate and separate the processing of data at different levels into different components. In this paper we present architecture-based refinement techniques for the design of multi-level secure systems. We discuss what security requirements must be satisfied through the refinement process, including when separation works and when it does not. The process oriented approach will lead to verified engineering techniques for secure systems, which should greatly reduce the cost of certification of those systems.**

Keywords: Architectural Refinement, Non-functional requirements, computer security.

I. INTRODUCTION

In recent years, security has been a growing concern in software engineering research, especially software architecture research. In IEEE/ANSI 830-1993, security is defined as one of the thirteen non-functional (or quality) requirements (NFRs) that must be included in the software requirements document. Other NFRs include performance, verification, acceptance, portability, reliability, maintainability, and safety [1]. NFRs refer to the whole software and thus cannot be presented in software architecture as components or functions offered by the system [2]. The overall software architecture and NFRs are closely related and should be studied together during architectural development. In the literature, much of the research focuses on how to satisfy NFRs such as reliability and performance [1], [3], [4], [5], [6], [7], instead of security. There has been some work on security architecture modelling [8], [9], but not on providing security architecture design guidance for architects. Banerjee et al. relate a software system's architecture and its trustworthiness (security, reliability, availability, fault-tolerance, and survivability) in general [10]. However, there is no additional detailed guidance.

The notion of enforcing security requirements at the architectural level is attractive because it allows security concerns to be recognized early in the development process and can be given sufficient attention in subsequent stages. By controlling system security during architectural refinement, we can decrease software production costs and speed up the time to market. This approach also enhances the role of the software architects by requiring that their decisions be not only for functional decomposition, but also for NFR fulfillment. In this paper, we present a set of architectural refinement patterns for system design architects to follow to achieve secure system architectures.

As some researchers have pointed out [2], [3], [5], effective refinement must be technology- and domain-specific. Also, Garlan [11] points out that refinement patterns must be explicit about what kinds of properties they are preserving in the refined design. In this paper, the type of systems within which our design refinement patterns are valid is a multi-level secure (MLS) system. We focus specifically on achieving confidentiality, not availability or integrity, for secure systems.

This paper is organized as follows: Section II introduces MLS systems and the concept of architecture; Section III proposes architectural refinement patterns that can be applied to MLS systems design. Section IV presents an example of MLS system design to illustrate the refinement patterns we proposed. Finally, we provide a discussion in Section V and conclusion and future work in Section VI.

II. BACKGROUND

A. MILS, MLS, MSLS and SLS

Traditionally, the military model of a secure system includes the concept of multi-level security (MLS). The idea behind this concept is that the system will be processing data items that are classified at different levels of security, and the security requirements that prevent the transfer of high-level classified information into low-level objects must be preserved. Therefore, we define an MLS system as one that must be certified to process and output co-mingled data at multiple classification levels. Classic security models, such as the Bell-LaPadula (BLP) model [12], have been used to specify the secure behavior of such MLS systems. Given a set of subjects, each with a clearance level, and a set of objects, each with a classification level, the BLP model requires that information does not flow downward by imposing the following requirements.

The Simple Security Property. A subject is allowed a read access to an object only if the subject's clearance level is identical to or higher than the object's classification level.

The *-Property. A subject is allowed a write access to an object only if the subject's clearance level is identical to or lower than the object's classification level.

[1]Contact Author: Jim Alves-Foss, Center for Secure and Dependable Systems, University of Idaho 83844-1008, jimaf@uidaho.edu, ph: 1-208-885-5196, fx: 1-208-885-9052

Report Documentation Page

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **2006**	2. REPORT TYPE	3. DATES COVERED **00-00-2006 to 00-00-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Architecture-Based Refinements for Secure Computer Systems Design	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Idaho,Center for Secure and Dependable Systems,PO Box 441008,Moscow,ID,83844-1008**	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**12**	

In this paper, we use security level to represent both the classification level and the clearance level.

The problem with full MLS systems is that they must be rigorously analyzed for security before they can be certified. Every portion of the MLS system must be analyzed to ensure that it properly handles labelled data and that there is no possible violation of the security requirements. Even with a trusted computing base (TCB) architecture or reference monitor [13], there is often too much to evaluate.

The Multiple Independent Levels of Security/Safety (MILS) architecture was developed to resolve the difficulty of certification of MLS systems by separating out the security mechanisms and concerns into manageable components [14]. These components are classified based on the way they process data:

- **SLS** Single-Level Secure component that only processes data at one security level.
- **MSLS** Multiple Single-Level Secure component that processes data at multiple security levels but always maintains separation between classes of data. An MSLS process or device separates the data into independent streams with no communication between streams. A device that processes messages one at a time (such as an I/O device driver) may be such a device.
- **MLS** Multi-Level Secure component that deals with data at multiple security levels and transforms the data from one level to another according to internal MLS security requirements of this component. Because of the potential seriousness of violating its security requirements, an MLS component requires the highest level of scrutiny and verification. Typically this can be a device that will downgrade information from a higher level of security to a lower level through either filtering or the application of encryption technology.

A MILS system isolates processes into partitions, which define a collection of data objects, code and system resources. These individual partitions can be evaluated separately, if the MILS architecture is implemented correctly. This divide-and-conquer approach will exponentially reduce the proof effort for secure systems. The MILS architecture is MLS if it enforces the MLS security requirements to regulate the communication between the applications and the resources, which is application-specific and not part of the MILS architecture.

B. Software Architecture

In the literature, a software architecture is usually described by components, connectors, and architectural configurations [15]. The following concepts are derived from Medvidovic and Taylor [15].

A *component* in an architecture is a unit of computation or a data store. They may be as small as a single procedure or as large as an entire application. Each component may require its own data or execution space, or it may share them with other components. The features of a component include interfaces, semantics, constraints, NFRs, etc.

*Connector*s are architectural building blocks used to model interactions between components and rules that govern those interactions. They may be message routing devices, shared variables, buffers, dynamic data structures, client-server protocols, pipes, SQL links between a database and an application, and so on. Connectors are characterized by their interfaces, semantics, constraints, NFRs, etc.

A *component's interface* is a set of interaction points between it and the external world. It specifies the services (messages, operations, and variables) a component provides to or that are required of other components in an architecture. In our approach, we augment the interface with security levels for each service.

A *connector's interface* is a set of interaction points between the connector and the components and other connectors attached to it. A connector does not perform any application-specific computations, and it exports as its interface those services it expects of its attached components. In our approach, we also augment the interface with security levels for each service.

*Architectural configuration*s are connected graphs of components and connectors that describe architectural structure. In this paper, we represent architectures as directed graphs. The graphs are integral to our refinement method, as they are used to represent the architectures throughout the refinement process and to specify pre- and post-conditions on each refinement step.

III. REFINEMENT PATTERNS

When designing secure systems, we do not want to separately construct abstract and concrete architectures and then prove that the concrete architecture satisfies the security properties of the abstract one. Instead, the concrete architecture should satisfy the security properties by construction, requiring no explicit proofs in its derivation. This can be accomplished through a series of small, local refinements, each of which involves the application of a secure refinement pattern. Then, the local refinements are combined to form the larger composite concrete architecture, which is guaranteed to correctly implement the abstract architecture, meaning that the security properties are not violated by the concrete architecture. In this way, the architecture can be refined to be more and more concrete until the implementation architecture is achieved.

In our paper, we do not focus on how to do functional requirements (FR) refinement, which has already been researched for many years with many approaches described in the literature (e.g., [16] and [17]). Instead, we focus on how to augment architectural refinement with security concerns. We present and justify informally a set of refinement patterns for refining components, connectors, and interfaces. The proposed patterns allow architects to decompose, aggregate, and eliminate components, connectors, and ports securely. In Section IV we apply these patterns to a simple example for illustration. Each refinement pattern is represented by a pair of directed graphs representing the pre- and post-condition of the refinement. In the graphs, a box represents a component,

an arrow represents a connector, and a dot represents a port. In the rest of this paper, we use ports to represent the interaction points of an interface.

A component/connector is said to be trusted if it is or will be designed to satisfy security requirements, while an untrusted component/connector has no security requirements. The security requirements associated with components/connectors describe not what they should do (e.g., send some data), but how they should do it (e.g., send data embedded in legal CORBA messages and with correct security labels). In our work we explicitly define two types of security requirements, intra-level requirements and inter-level requirements. Intra-level requirements are security requirements that are not related to information flow between security levels. For example, "Messages from Database must be CORBA reply messages" is such an intra-level requirement. This requirement is not on cross-level information flow, but on message types. Inter-level requirements are security requirements that are related to cross-level information flows. For example, "A top-secret message cannot flow to a secret component unless it is downgraded" is such an inter-level requirement. An SLS component only has intra-level requirements. An MSLS component, however, has not only intra-level requirements for each security level, it must also have an MSLS inter-level requirement that there is no cross-level information flow. An MLS component may have intra-level requirements for each security level and will have some MLS inter-level requirements to regulate cross-level information flow. In the final concrete architecture, if a connector has some security requirements, they must be application-independent.

Definition 1. (Secure Refinement) We say that a refinement on a part of the architecture (on components, connectors, ports, or a combination of these entities) is secure if and only if the new security requirements on this part after the refinement do not violate the original security requirements.

The refinement may add some new requirements, either functional or security requirements, but the new added requirements should not violate the original security requirements. This means that the security requirements after the refinement should be as strict as or stricter than those before the refinement.

In the following subsections, we present a basic set of refinement patterns for decomposition, aggregation, and elimination of components, connectors, and ports. We informally justify that these refinement patterns are secure if the pre- and post-conditions of the refinements are satisfied.

A. Decomposition Patterns

In the refinement process, it is typical to divide an architectural entity into smaller parts through decomposition. Decomposition can be applied to components, connectors, and ports.

1) Component Decomposition Patterns: A component can be decomposed into two components that are composed through product, cascade, or feedback [18], [19]. These three types of component decomposition are depicted in Figs. 1, 2,
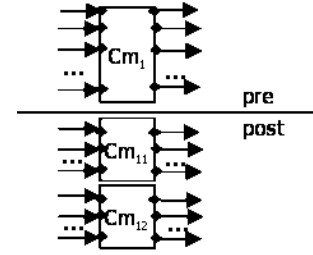

Fig. 1. Component Decomposition Product Pattern

and 4, respectively. Depending on the security requirements of the initial component, Cm_1, there are different possible requirements of the subcomponents, Cm_{11} and Cm_{12}. We discuss these in detail in this section and summarize them in Table I. Specifically, in Table I we list the initial component type (SLS, MSLS or MLS) as a pre-condition and then the possible refinements to subcomponent types as a post-condition, based on three different architectural decomposition approaches.

In addition to the post-conditions listed in Table I, there is an additional post-condition for all refinement patterns: services and security levels associated with each port connected to the environment are the same before and after the refinement. We will not repeat this condition when we elaborate pre-conditions and post-conditions of each secure refinement pattern.

Product Pattern. The first component decomposition pattern we discuss is Product (Fig. 1). In this pattern, Cm_1 is constructed by the parallel composition of Cm_{11} and Cm_{12}.

Product Condition 1 (SLS): If component Cm_1 is an SLS component, it can be securely refined into a product of Cm_{11} and Cm_{12} when both Cm_{11} and Cm_{12} are SLS components and where Cm_{11} and Cm_{12} together enforce the intra-level requirements of Cm_1.

Product Condition 2 (MSLS): If component Cm_1 is an MSLS component, there are three different secure refinements into the two SLS components, Cm_{11} and Cm_{12}.

1) The refinement is secure when Cm_{11} and Cm_{12} are both SLS components and each component enforces intra-level requirements for one security level, where Cm_1 is an MSLS component with just two different security levels.

2) The refinement is secure when one of the components (e.g., Cm_{11}) is an SLS component while another (e.g., Cm_{12}) is an MSLS component. In this case, usually Cm_1 is an MSLS component with more than two different security levels, Cm_{11} enforces intra-level requirements for one security level, and Cm_{12} enforces not only intra-level requirements for other security levels but also the inter-level requirement that there is no cross-level information flow.

3) The refinement is secure when both Cm_{11} and Cm_{12} are MSLS components. In one case, Cm_{11} or Cm_{12} enforces part of the intra-level security requirements in Cm_1 for all security levels in Cm_1 and enforces

TABLE I
SECURE COMPONENT DECOMPOSITION PATTERNS

Decomposition Approach	Pre-condition Cm_1	Cm_{11}			Cm_{12}		
		SLS	MSLS	MLS	SLS	MSLS	MLS
Product	SLS	√			√		
	MSLS	√			√		
		√				√	
			√			√	
	MLS	√					√
			√				√
				√			√
Cascade	SLS	√			√		
				√*			√*
	MSLS		√			√	
				√*			√*
	MLS	√					√
				√	√		
			√				√
				√		√	
Feedback	SLS	√			√		
				√*			√*
	MSLS		√			√	
				√*			√*
	MLS	√					√
				√	√		
			√				√
				√		√	
				√			√

* Extra inter-level requirements added in these cases

the inter-level security requirement that there is no cross-level information flow. Otherwise Cm_{11} or Cm_{12} enforces all intra-level security requirements in Cm_1 but for only some of the security levels (more than one), and enforces an inter-level requirement that there is no cross-level information flow.

Product Condition 3: If component Cm_1 is an MLS component, there are three secure refinements into two components, Cm_{11} and Cm_{12}.

1) The refinement is secure when one of the components (e.g., Cm_{11}) is an SLS component while the other (e.g., Cm_{12}) is an MLS component. In this case, Cm_{11} enforces the intra-level requirements for one security level, while Cm_{12} enforces the intra-level requirements for other security levels, and the inter-level requirements in Cm_{12} do not violate those in Cm_1.
2) The refinement is secure when one of the components (e.g., Cm_{11}) is an MSLS component while the other (e.g., Cm_{12}) is an MLS component. In this case, Cm_{11} enforces the intra-level requirements for more than one security level and an inter-level requirement to prohibit cross-level information flow; Cm_{12} enforces the intra-level requirements for the other security levels, and the inter-level requirements in Cm_{12} do not violate those in Cm_1.
3) The refinement is secure when both Cm_{11} and Cm_{12} are MLS components. In one case, Cm_{11} or Cm_{12} enforces part of the intra-level requirements in Cm_1 for all security levels in Cm_1 and enforces inter-level requirements in $Cm1$. Otherwise both Cm_{11} and Cm_{12} enforce all intra-level security requirements in Cm_1 but each for only some of the security levels (more than one), and the inter-level requirements in each component do not violate those in Cm_1.

In all of the product decomposition cases, we decompose the original component into two sub-components. The decomposition can separate our security concerns by separating the processing of difference security levels. When this happens, we get a more secure system. However, sometimes the decomposition does not separate security levels. In that case we need to still enforce inter-level security requirements as discussed above. In either case, the security requirements of the subcomponents, when composed, need to satisfy the requirement of the main component.

Cascade Pattern. The second decomposition pattern is Cascade (Fig. 2). In this pattern, Cm_1 is constructed through serial composition of Cm_{11} and Cm_{12}.

Cascade Condition 1: If component Cm_1 is an SLS component, there are two different secure refinements.

1) The refinement is secure when both Cm_{11} and Cm_{12} are SLS components and Cm_{11} and Cm_{12} together enforce the intra-level security requirements of Cm_1.
2) The refinement is secure when both Cm_{11} and Cm_{12} are MLS components. In this case, the intra-level requirements can be enforced by either Cm_{11} or Cm_{12} or by the two components together, and both need to enforce additional inter-level requirements that do not allow infomration flow between levels. Such a refinement
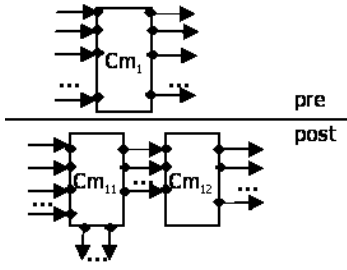
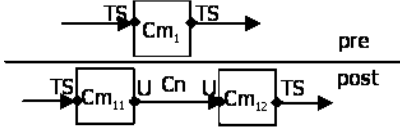Fig. 2. Component Decomposition Cascade Pattern

Fig. 3. Component Decomposition Cascade Pattern Example

could occur if one of these two MLS components is an up-grader while the other is a down-grader. This means that one component enforces an inter-level requirement to up-grade while the other enforces an inter-level requirement to down-grade. A typical example is shown in Fig. 3. In this example, Cm_1 is an abstract SLS communication component that takes in messages at Top Secret (TS) level and transfers the messages to output them also at TS. A possible refinement of Cm_1 would involve encrypted communication over an insecure channel (connector Cn). We could have Cm_{11} be an encryption device which takes in TS messages, encrypts them with a key for TS messages, then outputs Unclassified (U) messages, and Cm_{12} be a decryption device which takes in U messages, decrypts them with TS keys, then outputs TS messages. Obviously, Cm_{11} and Cm_{12} are both MLS components and with their composition, the transfer of TS messages is secure, with no message leakage to lower levels, as long as they support the new inter-level requirements.

Cascade Condition 2: If component Cm_1 is an MSLS component, there are two different secure refinements.

1) The refinement is secure when both Cm_{11} and Cm_{12} are MSLS components. Cm_{11} and Cm_{12} together enforce the intra-level security requirements of Cm_1, and each of them has an inter-level requirement to prohibit cross-level information flow.
2) The refinement is secure when both Cm_{11} and Cm_{12} are MLS components. In this case, the intra-level requirements can be enforced by either Cm_{11} or Cm_{12} or by the two components together. Similar to when Cm_1 is an SLS component, such a refinement could occur if one of these two MLS components is an up-grader while the other is a down-grader. Both subcomponents must satisfy the new inter-level requirements that state they do not violate the original security requirement.
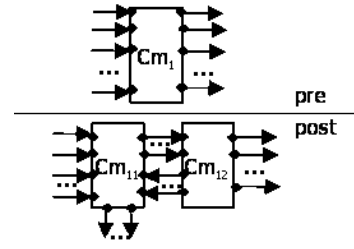

Fig. 4. Component Decomposition Feedback Pattern

Cascade Condition 3: If component Cm_1 is an MLS component, there are five different secure refinements.

1) The refinements are secure when Cm_{11}/Cm_{12} is an SLS component while Cm_{12}/Cm_{11} is an MLS component. In these two cases, components Cm_{11} and Cm_{12} together enforce the intra-level requirements of Cm_1, and Cm_{12}/Cm_{11} has MLS inter-level requirements which do not violate those of Cm_1.
2) The refinements are secure when Cm_{11}/Cm_{12} is an MSLS component and Cm_{12}/Cm_{11} is an MLS component. In these two cases, Cm_{11} and Cm_{12} together enforce the intra-level requirements of Cm_1, and the MLS requirements of Cm_{12}/Cm_{11} do not violate the MLS inter-level requirements of Cm_1.
3) The refinement is secure when both Cm_{11} and Cm_{12} are MLS components. In this case, Cm_{11} and Cm_{12} together enforce the intra-level security requirements of Cm_1, and the MLS inter-level requirements of Cm_{11} and Cm_{12} do not violate those of Cm_1.

Feedback Pattern. The final decomposition pattern is Feedback (Fig. 4). In this pattern, Cm_1 is constructed by two communication components Cm_{11} and Cm_{12}.

The three secure refinement cases summarized in Table I are similar to those for Cascade pattern and are omitted here to save space. However, it is important to note that some security properties are not preserved under feedback composition [18], [19], [20]. In Section V, we discuss feedback composition further.

2) Connector and Port Decomposition Patterns: In addition to component decomposition, we need to discuss connector and port decomposition as well.

As discussed in Section II, in the final concrete architecture, a connector does not perform any application-specific computations. Therefore, during the architectural refinement, for any connector with application-specific security requirements on it, we should refine this connector using ConnDec pattern (Fig. 5). In pattern ConnDec, connector Cn is decomposed into connectors Cn_1 and Cn_2 and component Cm. After the ConnDec refinement, all application-specific security requirements of Cn are put on component Cm and no application-specific security requirements are put on connectors Cn_1 and Cn_2. Effectively, Cm is a guard that enforces the application-specific security requirements of Cn. The connectors Cn_1 and Cn_2 and the component Cm together should enforce the
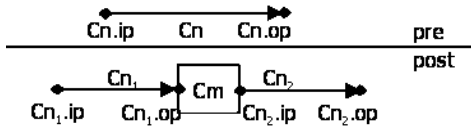
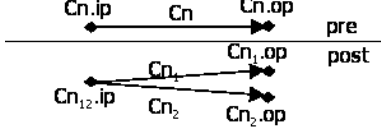Fig. 5.   Connector Decomposition ConnDec Pattern

Fig. 6.   Port Decomposition PortDec1 Pattern

security requirements on Cn.

Two other post-conditions are: $Cn_1.ip$ has the same services and security levels as those of $Cn.ip$ while $Cn_2.op$ has the same services and security levels as those of $Cn.op$; $Cn_1.op$ has the same security levels as $Cn_1.ip$, and $Cn_2.ip$ has the same security levels as $Cn_2.op$.

When a connector has application-independent security requirements on it, it is secure to refine this connector into a serial composition of two connectors if these two connectors together enforce the application-independent security requirements (not pictured here).

For port decomposition, we decompose a port into multiple ports, splitting the services and security levels of the original port to associate them with the new ports. When a port of a component decomposes, nothing changes in the component and no security requirements will be violated. For the connector to this port, in one case, such a port decomposition causes the connector to decompose also (Fig. 6). In this case, we need to split the security requirements of the connector Cn to put them into Cn_1 and Cn_2. Possible inter-level requirements on Cn_1 and Cn_2 should not violate the inter-level requirements on Cn.

In another case, a port decomposition does not cause the connector to decompose (Fig. 7). In this case, connectors Cn_1 and Cn_2 remain the same, meaning the security requirements in these two connectors remain the same.

B. Aggregation Patterns

When refining an architecture, we may find it is necessary to merge two more abstract entities into a single lower level entity. Aggregation can be applied to components, connectors,
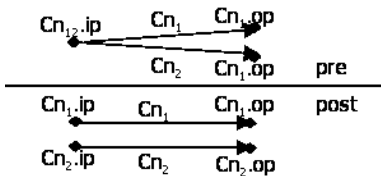

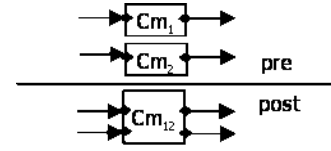Fig. 7.   Port Decomposition PortDec2 Pattern

Fig. 8.   Component Aggregation ComAgg Pattern

and ports.

If two components are not connected directly, when we merge them as one component, we say we aggregate them. If two components are connected directly, we can consider them as part of a bigger component, but not an aggregation.

ComAgg Pattern. The component aggregation pattern we provide in this paper is ComAgg (Fig. 8). The six secure refinement cases are summarized in Table II.

TABLE II
COMPONENT AGGREGATION PATTERNS TABLE

Pre-condition		Post-condition		
Cm_1	Cm_2	Cm_{12}		
		SLS	MSLS	MLS
SLS	SLS	$\checkmark$		
			$\checkmark^*$	
SLS	MSLS	$\checkmark^*$		
SLS	MLS			$\checkmark^*$
MSLS	MSLS		$\checkmark$	
MSLS	MLS			$\checkmark$
MLS	MLS			$\checkmark$

* Extra inter-level requirements added in these cases

ComAgg Condition 1: If both Cm_1 and Cm_2 are SLS components, after aggregation, Cm_{12} could be an SLS or MSLS component. When Cm_1 and Cm_2 have the same security level, Cm_{12} is an SLS component that has the intra-level requirements of Cm_1 and Cm_2; when Cm_1 and Cm_2 have different security levels, Cm_{12} should be an MSLS component that has not only the intra-level requirements of Cm_1 and Cm_2, but also a new MSLS inter-level requirement to prohibit cross-level information flow.

ComAgg Condition 2: If one of the components, Cm_1 or Cm_2, is an SLS component and the other is an MSLS component, Cm_{12} is an MSLS component that has not only the intra-level requirements of Cm_1 and Cm_2, but also an MSLS inter-level requirement to prohibit cross-level information flow.

ComAgg Condition 3: If either component Cm_1 or Cm_2 is an SLS component (e.g., Cm_1) and the other is an MLS component (e.g., Cm_2), Cm_{12} should be an MLS component that has the intra-level requirements of Cm_1 and Cm_2. The MLS inter-level requirements of Cm_{12} should not violate those of Cm_2.

ComAgg Condition 4: If both Cm_1 and Cm_2 are MSLS components, Cm_{12} must be an MSLS component that enforces all intra-level requirements of Cm_1 and Cm_2 and has an MSLS inter-level requirement to prohibit any cross-level information flow.

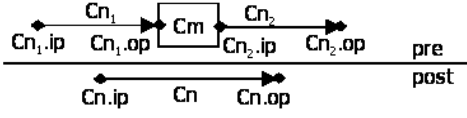*ComAgg Condition 5:* If either Cm_1 or Cm_2 is an MSLS component while the other is an MLS component, Cm_{12}

Fig. 9. Component and Connector Aggregation ComConnAgg Pattern

Fig. 10. Port Aggregation PortAgg1 Pattern

Fig. 11. Port Aggregation PortAgg2 Pattern

should be an MLS component that has all intra-level requirements of Cm_1 and Cm_2. The MLS inter-level requirements of Cm_{12} should not violate those of Cm_1 or Cm_2. If there is any conflict between the MLS inter-level requirements and the MSLS inter-level requirements of the original components, (e.g., in the MLS inter-level requirements, there is cross-level information flow between TS and S, but in the MSLS inter-level requirements, there should not be any cross-level information flow) we require the stricter requirement. That means in Cm_{12} we follow the MSLS inter-level requirement instead of the MLS inter-level requirement when conflict occurs.

ComAgg Condition 6: If both Cm_1 and Cm_2 are MLS components, Cm_{12} should be an MLS component that enforces all intra-level requirements of Cm_1 and Cm_2, and the MLS inter-level requirements of Cm_{12} should not violate those of Cm_1 and Cm_2.

It is important to note that in ComAgg, though the two components do not interact directly with each other, they both interact with their environment, and in some cases, their aggregation may cause a subtle situation that needs to be examined closely, as will be discussed in Section V.

ComConnAgg Pattern. We propose an aggregation pattern ComConnAgg (Fig. 9). In this pattern, connectors Cn_1 and Cn_2 and component Cm are aggregated to a new connector Cn. This aggregation is secure if Cn is trusted to enforce all the security requirements of Cn_1, Cn_2, and Cm.

Also, two connectors composed serially can be aggregated to one connector if the new connector does not violate any security requirements of the two original connectors.

PortAgg1 and PortAgg2 Patterns. For port aggregation, we aggregate multiple ports into one, merging the services and security levels of the original ports to associate them with the new port. When the ports of a component are aggregated, nothing changes in the component and security is not violated. For the connectors to these ports, in one case, such a port aggregation causes the connectors to aggregate as well (Fig. 10). In this case, Cn_{12} should have all the security requirements of Cn_1 and Cn_2. Only when $Cn_{12}.ip$, $Cn_1.op$, and $Cn_2.ip$ have the same security level will there be no inter-level requirements on Cn_{12}; otherwise, there should be inter-level requirements on Cn_{12}, and these should not violate the inter-level requirements of Cn_1 and Cn_2.

In another case, ports aggregation does not cause connectors with these ports to aggregate (Fig. 11). In this case, after the port aggregation, connectors Cn_1 and Cn_2 remain the same, meaning all security requirements of these two connectors remain the same.
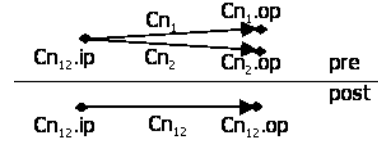
C. Elimination Patterns

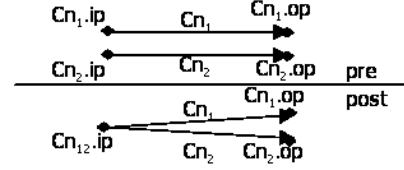Simply stated, for any connector with no semantics, meaning no functionality is provided by it, we can securely eliminate this connector. Elimination of a component is secure if the component has no connection to other components. Also, elimination of a port is secure if it has no connection to its environment.

IV. A MILS APPLICATION DESIGN EXAMPLE

In this section, we present an example of a MILS application system to illustrate how to refine a system design according to the refinement patterns described in the previous section. The system is simple enough to avoid complexity for the purposes of discussion but sufficient to illustrate the concepts discussed. The specifications for the system are: A distributed, MLS secure (suppose there are two security levels used, TS and S), MILS application system that allows users with different security levels to store and retrieve data on an MLS database using legal CORBA messages (i.e., legal means read, write methods for users and reply method for database). Assume user U_1 is at TS level while user U_2 is at S level and the database DB is remotely accessible by both U_1 and U_2. The functional requirements of this application are: store and retrieve data. The security requirements are: 1. inter-level requirements: store/retrieve data obeying BLP model; 2. intra-level requirements: all messages sent should be legal CORBA messages.

The abstract architecture of this system is depicted in Fig. 12.

In the abstract architecture, connector Cn_{U_1-DB} has the FR that it forwards messages from U_1 to DB, located at different computers. The security level associated with the ports of
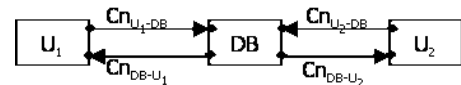

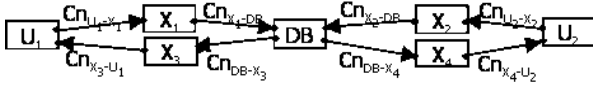Fig. 12. The Abstract Architecture of the Application Example

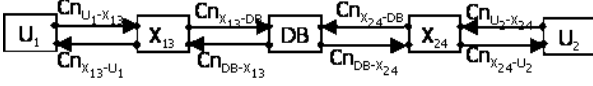Fig. 13. An Architecture of the Example after Step 1: ConnDec

Fig. 14. An Architecture after Step 2: ComAgg

Fig. 15. An Architecture after Step 3: ComAgg

Fig. 16. Part of Architecture after Step 4: Feedback

this connector is TS. The security requirements of Cn_{U_1-DB} are listed in Table III, No. 1 and 2. Connectors Cn_{U_2-DB}, Cn_{DB-U_1}, and Cn_{DB-U_2} have similar functionality and security requirements.

We now illustrate several refinement steps of the application system design to obtain a secure concrete system architecture.

Step 1. The abstract architecture can be first refined by applying the connector decomposition pattern ConnDec to the connector Cn_{U_1-DB}. The original connector is decomposed into connector $Cn_{U_1-X_1}$, component X_1, and connector Cn_{X_1-DB} (Fig. 13). According to the post-conditions of pattern ConnDec, we place application-specific security requirements on component X_1 and make the connectors $Cn_{U_1-X_1}$ and Cn_{X_1-DB} application-independent. After the refinement, $Cn_{U_1-X_1}$ has the FR that it forwards messages between different partitions in MILS architecture, and the NFR that the security level associated with the ports of this connector remains TS. The FR of connector Cn_{X_1-DB} is the same as that of $Cn_{U_1-X_1}$, and the security level associated with the ports of this connector remains TS. Component X_1 has the security requirements in Table III, No. 7 and 8. Also, X_1 has the functionality to forward messages between different computers. Similarly, the decomposition pattern ConnDec is applied to the connectors Cn_{U_2-DB}, Cn_{DB-U_1}, and Cn_{DB-U_2}, and the new architecture is achieved (Fig. 13).

Here, we can say that this is a big-step refinement which includes multiple mini-steps. A mini-step refinement applies one refinement pattern to a component, connector, or interface, while a big-step is a set of mini-steps applied to the architecture refinement sequentially.

Step 2. We take the architectural refinement further by applying component aggregation pattern ComAgg to the architecture of Fig. 13, resulting in the architecture in Fig. 14.

In this step of refinement, the component aggregation pattern ComAgg is applied to X_1 and X_3. After the refinement, the security requirements of component X_{13}, No. 13 and 14 in Table III, do not violate the security requirements of X_1 and X_3. Also, X_{13} has the FR of forwarding messages between different computers. Similarly, the component aggregation pattern ComAgg is applied to X_2 and X_4 to get a component X_{24}. The services and security levels of all ports remain the same as those in Fig. 13.

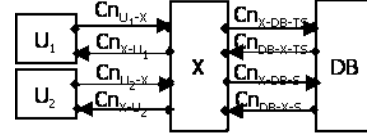Step 3. Further, we apply ComAgg pattern to the previous architecture to aggregate components X_{13} and X_{24}, and the new architecture is in Fig. 15.
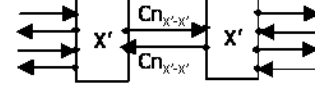
After the refinement, component X has the security requirements No. 17 and 18 in Table III which do not violate the security requirements of X_{13} and X_{24}. Also, component X has the FR of forwarding messages between different computers. Services and security levels associated with all ports remain the same as those in Fig. 14.

Step 4. Now, we apply component decomposition pattern Feedback to decompose component X to a feedback composition of X' and X' shown in Fig. 16.

In this step of refinement, component X' has the security requirements No. 19 and 20 in Table III, which also do not violate the security requirements of X. X' has new MLS inter-level requirements, No. 21, to downgrade TS/S messages to U messages by encrypting them with a TS/S key before forwarding, and to upgrade U messages to TS/S by decrypting them with a TS/S key before forwarding. These new MLS inter-level requirements do not violate the security requirements of X. Connectors $Cn_{X'-X'}$ transfer messages between different computers. These connectors are application-independent. The security level associated with the ports of these connectors is U.

Step 5. We continue refining component X' in Fig. 16 by applying component decomposition pattern Feedback to decompose X' into a feedback composition of X'' and a component named $TNIU$ as shown in Fig. 17. The component $TNIU$ has the MLS inter-level requirements, No. 24 in Table III, to downgrade/upgrade all messages it receives before forwarding them. Component X'' has the security requirements No. 22 and 23 in Table III, which do not violate the security requirements No. 19 and 20 of X'. Connectors $Cn_{X''-TNIU}$ and $Cn_{TNIU-X''}$ can transfer messages using direct process calls.

Step 6. Now, the component X'' in Fig. 17 can be decomposed into components MMR and GG by applying component decomposition pattern Feedback as shown in Fig.
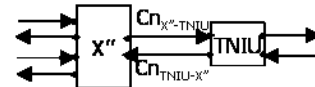

Fig. 17. Part of the Architecture after Step 5: Feedback

TABLE III

APPLICATION REQUIREMENTS ENFORCEMENT TABLE

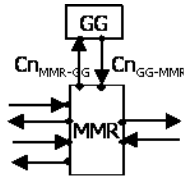| Com/Conn	No.	Intra-level Requirements	Inter-level Requirements
Cn_{U_1-DB}	1	Messages are legal CORBA messages.	
	2		TS level, read and write according to BLP.
Cn_{U_2-DB}	3	Messages are legal CORBA messages.	
	4		S level, read and write according to BLP.
Cn_{DB-U_1}	5	Messages are legal CORBA messages.	
Cn_{DB-U_2}	6	Messages are legal CORBA messages.	
X_1	7	Messages are legal CORBA messages.	
	8		U_1 read/write TS messages and read S messages.
X_2	9	Messages are legal CORBA messages.	
	10		U_2 read/write S messages and write TS messages.
X_3	11	Messages are legal CORBA messages.	
X_4	12	Messages are legal CORBA messages.	
X_{13}	13	Messages from U_1 to DB are legal CORBA messages; messages from DB to U_1 are legal CORBA messages.	
	14		U_1 read/write TS data and read S data.
X_{24}	15	Messages from U_2 to DB are legal CORBA messages; messages from DB to U_2 are legal CORBA messages.	
	16		U_2 read/write S data and write TS data.
X	17	Messages from U_1 and U_2 to DB are legal CORBA messages; messages from DB to U_1 and U_2 are legal CORBA messages.	
	18		U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
X'	19	Messages from U_1 and U_2 to DB are legal CORBA messages; messages from DB to U_1 and U_2 are legal CORBA messages.	
	20		U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
	21		Downgrading TS/S messages to U by encrypting with TS/S key; upgrading U messages to TS/S by decrypting with TS/S key.
X''	22	Messages from U_1 and U_2 to DB are legal CORBA messages; messages from DB to U_1 and U_2 are legal CORBA messages.	
	23		U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
$TNIU$	24		Downgrading TS/S messages to U by encrypting with TS/S key; upgrading U messages to TS/S by decrypting with TS/S key.
MMR	25	Forwarding different types of messages to the correct type checking components.	
	26		Messages from different partitions must be labelled correctly with security levels of the partitions.
GG	27	All messages received must be checked to be legal CORBA messages.	
	28		U_1 read/write TS data and read S data; U_2 read/write S data and write TS data.
DB	29	Reply with legal CORBA messages.	
	30		Reply to requests with data at proper classification levels.

Fig. 18. Part of the Architecture after Step 6: Feedback

18.

In this step of refinement, the component MMR enforces the intra-level requirement No. 25 and the MSLS inter-level requirement No. 26 in Table III. The component GG, representing a GIOP Guard, enforces security requirements No. 27 and 28 in Table III. Connectors Cn_{MMX-GG} and Cn_{GG-MMX} can transfer messages using direct process calls, and the security levels associated with the ports of this connector are TS and S.

After all these steps, we take one additional refinement step by applying port aggregation patterns PortAgg2 and PortAgg1 sequentially to aggregate the ports and connectors between the MMR and DB to get the final concrete architecture shown in Fig. 19. The security levels associated with the ports of the connectors between MMR and DB are TS and S.
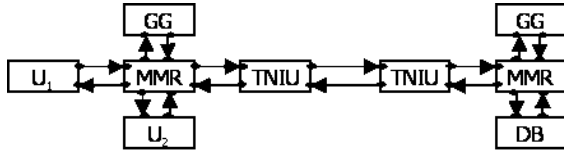
Fig. 19. Final Concrete Architecture of the Application Example
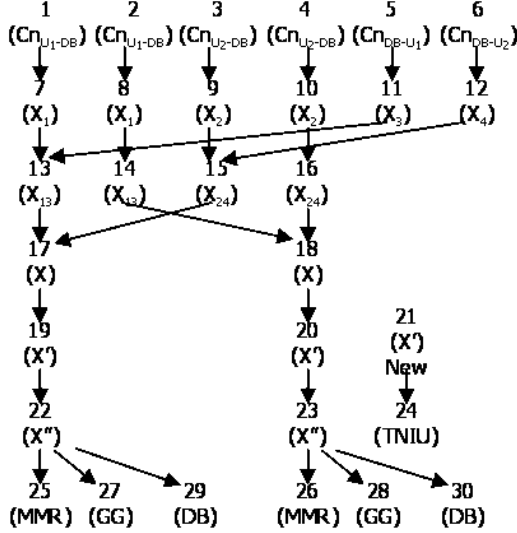

Fig. 20. Trace of Security Requirements Enforcement

Through the justification of each step of refinement, we can informally guarantee that this concrete architecture is a secure design of the application example, if components MMR, GG, $TNIU$, and DB are trusted to enforce all security requirements put on them during the refinement as in Table III.

The trace of how the original security requirements are enforced during the refinement process is presented in Fig. 20. The numbers in the figure represent the respective security requirements in Table III.

By walking through the design of this application example, we can see that our refinement patterns can be guidance for architects to apply them to each step of architecture design refinement and to justify each mini-step to reach a concrete architecture with no need for additional proofs.

V. DISCUSSION

In this paper, when we design a secure MLS system, we enforce intra-level requirements and prohibit any direct information flow that is not permitted by the inter-level requirements, but we do not address indirect information flow. If a system needs to achieve a stringent security property, such as generalized non-interference (GNI) [20], [18], [19], feedback composition of two components must be examined closely. As pointed out in the literature, in some cases, the direct feedback composition of two secure components could be insecure in terms of GNI.

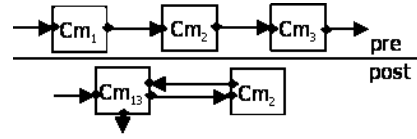In the refinement patterns we presented, the direct feedback

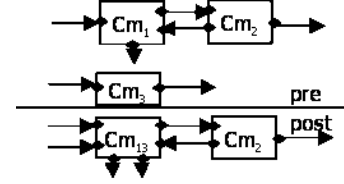Fig. 21. Feedback Composition of Two Components after Aggregation

Fig. 22. Possible Insecure Component Aggregation Example

composition of two components occurs in the component decomposition pattern Feedback. Alternatively, the feedback composition of two components could occur after applying the component aggregation ComAgg (Fig. 21), where components Cm_1 and Cm_3 aggregate to a component Cm_{13} and Cm_{13} is feedback composed with Cm_2 after the refinement.

In some cases, a feedback composition is secure before refinement, but after refinement, the composition could be insecure (Fig. 22). In this component aggregation, the direct feedback composition of Cm_1 and Cm_2 is secure before the aggregation; however, if Cm_1 and Cm_3 deal with different security levels, the direct feedback composition of Cm_{13} and Cm_2 could be insecure when these two components are aggregated.

No matter what causes the feedback composition of two non-SLS components, the architect should not only put MSLS requirements or MLS requirements on the components, but also refer to secure composition approaches [20], [19] to check and eliminate any unpermitted indirect cross-level information flow. For example, one approach is to add a buffer in the feedback connection between these two components. Only when Cm_1 and Cm_2 are both SLS components and at the same security level do architects not need to consider this subtle situation.

In component decomposition pattern Cascade, when Cm_1 is an MSLS component, either Cm_{11} or Cm_{12} could be an MSLS component while the other is an SLS component (Fig. 23). However, we do not list these post-conditions in Table I since such a decomposition can be achieved by applying a set of refinement patterns sequentially (Fig. 24). We can consider Cascade(MSLS→SLS+MSLS) as a pattern of big-
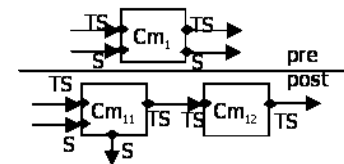

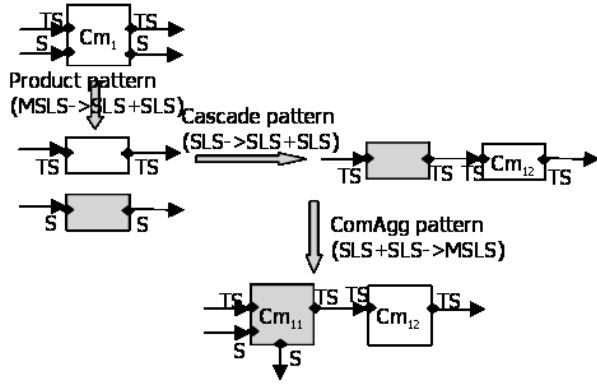Fig. 23. A Cascade Pattern (MSLS→SLS+MSLS)

Fig. 24.   Applying a Set of Mini-step Refinements

step refinement which can be achieved through applying a set of mini-steps.

For component decomposition pattern Feedback, when Cm_1 is an MSLS component, either Cm_{11} or Cm_{12} could be an MSLS component while the other is an SLS component. We do not list these post-conditions in Table I for similar reason as for pattern Cascade.

VI. Conclusions and Future Work

In this paper, we present a set of refinement patterns for components, connectors, and interfaces in architectures. With informal justification of these patterns, we provide a framework for secure system architecture design.

In subsequent work, we will formally verify that these refinement patterns are secure by mathematically modelling these patterns and providing mathematics theorems as criteria for secure refinement.

We will work on deciding a necessary and sufficient set of basic refinement patterns on which other refinement patterns can be built. For example, the Cascade(SLS→SLS+SLS) is such a pattern, but Cascade(MSLS→SLS+MSLS) discussed in Section V is not. We call this set the Core set, or Level-0 set. New patterns can be built on patterns in Level-0, and a Level-1 set consists of the new patterns. Intuitively, the architects can build their own patterns in any Level-n set based on the patterns in Level-0 to Level-(n-1) sets. The verification of patterns in Level-n can build on the verification of patterns in lower level sets, which can reduce the verification efforts.

Finally, we will work on formalizing the secure architectural refinement framework by using Unified Modeling Language (UML). Since UML is used during software development from early phases to detailed design, performing and presenting the architectural refinements of secure systems in UML will make them comprehensible to everyone involved. With a formal framework of secure architectural refinements in system design, we can perform solid components interaction analysis and verify that their composition results in desired behavior and security properties.

References

[1] N. S. Rosa, G. R. R. Justo, and P. R. F. Cunha, "A framework for building non-functional software architectures," in *Proc. 2001 ACM Symposium on Applied Computing (SAC'01)*, Las Vegas, Nevada, United States, 2001, pp. 141–147.

[2] V. Ambriola and A. Kmiecik, "Architectural transformations," in *Proc. 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, Ischia, Italy, 2002, pp. 275–278.

[3] K. S. Barber, T. Graser, and J. Holt, "Enabling iterative software architecture derivation using early non-functional property evaluation," in *Proc. 17th IEEE International Conference on Automated Software Engineering (ASE '02)*, 2002, p. 172.

[4] L. Chung, B. A. Nixon, and E. Yu, "An approach to building quality into software architecture," in *Proc. 1995 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'95)*, Toronto, Ontario, Canada, 1995, p. 13.

[5] M. Denford, J. Leaney, and T. O'Neill, "Non-functional refinement of computer based systems architecture," in *Proc. 11th IEEE International Conference and Workshop on the Engineering of Computer-based Systems (ECBS'04)*, 2004, p. 168.

[6] X. Franch and P. Botella, "Putting non-functional requirements into software architecture," in *Proc. of the 9th International Workshop on Software Specification and Design (IWSSD'98)*, 1998, p. 60.

[7] N. S. Rosa, G. R. R. Justo, and P. R. F. Cunha, "Incorporating non-functional requirements into software architectures," in *Proc. 15 IPDPS 2000 Workshops on Parallel and Distributed Processing (IPDPS'00)*, 2000, pp. 1009–1018.

[8] Y. Deng, J. Wang, J. J. P. Tsai, and K. Beznosov, "An approach for modeling and analysis of security system architectures," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 5, pp. 1099–1119, 2003.

[9] M. Moriconi, X. Qian, R. A. Riemenschneider, and L. Gong, "Secure software architectures," in *Proc. 1997 IEEE Symposium on Security and Privacy (SP'97)*, 1997, p. 84.

[10] S. Banerjee, C. A. Mattmann, N. Medvidovic, and L. Golubchik, "Leveraging architectural models to inject trust into software systems," in *Proc. 2005 Workshop on Software Engineering for Secure Systems Building Trustworthy Applications (SESS'05)*, St. Louis, Missouri, 2005, pp. 1–7.

[11] D. Garlan, "Style-based refinement for software architecture," in *Joint Proc. 2nd International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, San Francisco, California, United States, 1996, pp. 72–75.

[12] D. E. Bell and L. LaPadula, "Secure computer systems: Unified exposition and multics interpretation," *MITRE Technical Report, MITRE Corporation, Bedford Massachusetts*, vol. 2997, p. ref A023 588, 1976.

[13] J. P. Anderson, "Computer security technology planning study," Fort Washing, Pennsylvania, Tech. Rep., 1972.

[14] J. Alves-Foss, W. S. Harrison, P. Oman, and C. Taylor, "The MILS architecture for high-assurance embedded systems," *International Journal of Embedded Systems*, vol. 1, no. 1, Jan. 2005.

[15] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, no. 1, pp. 70–93, 2000.

[16] M. Moriconi, X. Qian, and R. A. Riemenschneider, "Correct architecture refinement," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 356–372, 1995.

[17] J. Philipps and B. Rumpe, "Refinement of pipe-and-filter architectures," in *Proc. World Congress on Formal Methods in the Development of Computing Systems (FM'99)-Volume I*, 1999, pp. 96–115.

[18] J. McLean, "A general theory of composition for a class of "possibilistic" properties," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 53–67, Jan. 1996.

[19] A. Zakinthinos, "On the composition of security properties," Ph.D. dissertation, University of Toronto, Mar. 1996.

[20] D. McCullough, "Noninterference and the composability of security properties," in *Proc. IEEE Symposium on Security and Privacy*, 1988, pp. 177–187.